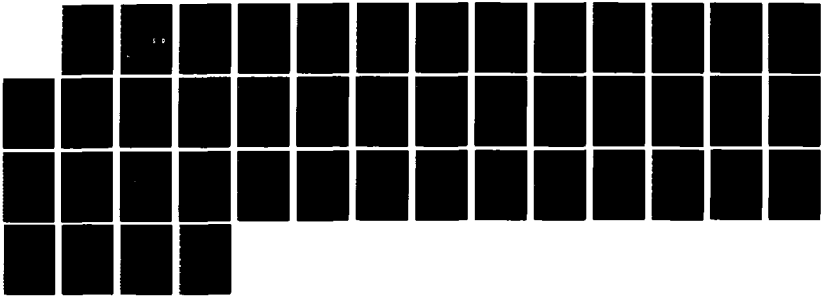
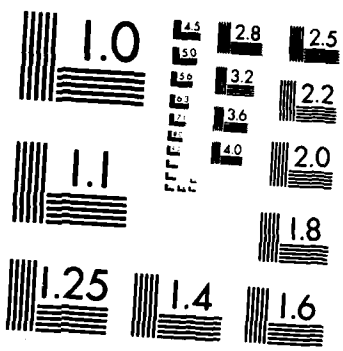
AD-A171 628 HIGH QUALITY 16 KBIT/S VOICE A/D IMPLEMENTATION(U) BOLT 1/1
BERANEK AND NEWMAN INC CAMBRIDGE MA J TIAO ET AL.
APR 86 BBN-6205 DCA100-84-C-0044

UNCLASSIFIED F/G 17/2 NL





BBN Laboratories Incorporated

A Subsidiary of Bolt Beranek and Newman Inc.

Report No. 6205

HIGH QUALITY 16 KBIT/S VOICE A/D IMPLEMENTATION

FINAL REPORT

J. Tiao, L. Cosell, K. Field, A.G. Derr,
W. Russell, M. Krasner, and V. Viswanathan

April 1986

Prepared for:

Defense Communications Agency

Report No. 6205

HIGH QUALITY 16 KBIT/S VOICE A/D IMPLEMENTATION

FINAL REPORT

J. Tiao, L. Cosell, K. Field, A. Derr, W. Russell, M. Krasner, and V. Viswanathan

April 1986

Prepared by:

BBN Laboratories Incorporated
10 Moulton Street
Cambridge, Massachusetts 02238

Prepared for:

Defense Communications Agency
Defense Communications Engineering Center
1860 Wiehle Avenue
Reston, VA 22090

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: BBN Report No. 6205	2. GOVT ACCESSION NO.: AD-A171628
	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): High Quality 16 KBit/s Voice A/D Implementation	5. TYPE OF REPORT & PERIOD COVERED: Final Report April 1984 - April 1986
	6. PERFORMING ORG. REPORT NUMBER: BBN Report No. 6205
7. AUTHOR(s): J. Tiao, L. Cosell, K. Field, A. Derr, W. Russell, M. Krasner, and V. Viswanathan	8. CONTRACT OR GRANT NUMBER(s): DCA100-84-C-0044
9. PERFORMING ORGANIZATION NAME AND ADDRESS: BBN Laboratories Incorporated, 10 Moulton Street, Cambridge, MA 02238	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Communications Agency, Contract Management Division, Code 680, Washington, D.C. 20305	12. REPORT DATE: April 1986
	13. NUMBER OF PAGES: 36
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Defense, for sale to the general public.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Speech coding, 16 kbit/s speech transmission, 2.4 kbit/s speech transmission, medium-bandwidth speech transmission, narrow-bandwidth speech transmission, adaptive predictive coding, digital voice terminal, real-time speech coder, portable voice communication unit.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the design and construction of prototype portable voice communication units and the implementation of the BBN robust 16 kbit/s adaptive predictive coding algorithm as a full-duplex real-time speech coder on these units. The report documents the hardware and software design and implementation efforts, and presents the results of a hardware production cost study. Work on algorithm simplification of the BBN 2.4 kbit/s harmonic deviations LPC speech coder is also described.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The authors would like to acknowledge the help of their colleagues G. Fedorkow, J. Goodhue, and D. Judelson. Also, the authors would like to thank J. Lambert and G. Moran of the Defense Communications Agency for their interest and encouragement during the course of this project.

1. INTRODUCTION AND EXECUTIVE SUMMARY

1.1 Project Goals

The overall objective of this project was the development of three prototype portable voice communication units. The original Statement of Work (SOW) specified that these units were to produce high quality speech at 2.4 kbits/s and toll quality speech at 16 kbits/s. The contract was subsequently modified to eliminate the 2.4 kbit/s requirement. The completed units meet the 16 kbit/s requirement through the use of the speech coding algorithm called the Adaptive Predictive Coder with Noise Shaping (APC-NS), which was developed by BBN under Contract No. DCA100-79-C-0037 [1, 2].

At 16 kbits/s, the units were required to provide toll quality speech under normal conditions, provide high quality speech with channel bit-error rates up to 1%, operate in an office noise environment, and tandem satisfactorily with LPC-10. (Toll quality speech is understood by the speech community to be speech that is natural sounding and contains no audible degradations (comparable in quality to Direct Distance Dialing (DDD) telephone speech). High quality speech is natural sounding but contains audible degradations.)

The units were to be designed to operate full-duplex in real-time. Each unit was to have: a unity overall processing gain; balanced differential 600 ohm input and output for tape recorder interface; a handset with a high quality dynamic microphone having 100-5000 Hz passband; analog interface input and output filters with 100-3200 Hz passband (for the APC-NS coder); and an EIA standard RS-232 modem interface. The SOW also called for a study of the potential production costs of the units.

1.2 Highlights of the Work

We have designed and implemented a single-board high-speed signal processing Voice A/D (VAD) device, capable of running a variety of real-time speech and signal processing applications. A 10-MHz Motorola MC68000 functions as the central controlling processor, with up to three TI TMS32010 processors providing computational power. A shared memory scheme is employed for interprocessor communication. The board has a high quality analog interface that supports telephone, dynamic microphone, and tape recorder I/O; a dedicated synchronous RS-232 port for transmission and receipt of digitally coded speech; and four programmable asynchronous/synchronous RS232 ports for communicating with other devices.

The 12" x 18" board has been packaged as a self-contained portable system. The packaging includes a high-quality handset, power supply, analog and digital device connectors, control switches, and LED indicators.

We have developed a 16-bit fixed-point version of BBN's 16 kbit/s APC-NS coder. The fixed-point coder does not differ significantly in output quality from the original floating-point coder; they both produce toll quality speech with no channel errors, and high quality speech with channel bit-error rates up to 1%. Using a data-driven real-time software design, we subsequently implemented this fixed-point APC-NS algorithm on the VAD hardware. Each VAD/APC-NS unit functions as one half of a two-unit full-duplex 16 kbit/s speech communication system.

Our real-time implementation required the use of only two of the three available TMS32010 processors on the VAD board. The APC-NS transmitter was implemented on one of the processors; it executes in about 80.8% of real time. The receiver was implemented on the other processor, and requires about 17.7% of real time. The MC68000 was used for interrupt processing and system control. This processing consumes about 41% of the available time. We have measured the overall system delay (not including channel transmission delay) introduced by our APC-NS real-time implementation to be about 115 ms.

We have demonstrated that the VAD/APC-NS coder produces toll quality speech

with no channel errors. We have used the government-provided channel error simulation box to introduce 1% channel errors; under this condition the coder produces high quality speech. In fact, the effect of 1% bit-errors is just barely audible. In addition, we have demonstrated that the coder performs satisfactorily in tandem with LPC-10.

1.3 Overview of the Report

In the body of this report we present a description of our software implementation of the APC-NS coder (Section 2), and our preliminary work on the implementation of the 2.4 kbit/s HDV coder (Section 3). We then report on our hardware design and implementation effort, our real-time software design and implementation, and our hardware/software integration work (Section 4). Finally, we present the results of our production cost study (Section 5).

2. 16 KBIT/S APC-NS CODER

2.1 Algorithm Description

We developed and optimized the APC-NS system in 1980 for real-time full-duplex operation, with the goals of producing good quality speech under transmission channel bit error rates of up to 1%, producing toll quality speech in office noise environments, and producing good quality speech in Air-Borne Command Post (ABCP) noise environments [1]. This real-time APC-NS system was also designed to perform well in tandem with the current Department of Defense (DOD) standard 2.4 kbit/s LPC-10 coder.

This system was implemented as a real-time full-duplex speech communication terminal using the CSPI MAP-300, which is a multi-processor asynchronous signal processing computer, capable of about 12 MFLOPS [2, 3].

A detailed description of the APC-NS algorithm is given in [1]. The algorithm may be summarized as follows. In the transmitter, the analog input speech is highpass filtered at 100 Hz, lowpass filtered at 3.2 kHz, sampled at 6.67 kHz, and divided into frames of 33.75 ms duration (225 samples). Each frame of speech is preemphasized using a fixed single-zero filter and encoded using the APC encoder, to produce the quantized residual samples for that frame. The APC encoder employs: 3-tap pitch prediction and 6-pole spectral prediction to obtain the residual; forward-adaptive quantization of the residual samples using 2 bits/sample; and pole-zero spectral shaping of the quantization noise to reduce its perception at the coder output. The parameters of the spectral and pitch predictors and of the adaptive quantizer are quantized, coded, partially error-protected using a total of 11 Hamming (7,4) codewords, and transmitted over the channel along with the residual samples, which are encoded using a folded-binary code.

In the receiver, the decoded residual samples are applied to the input of a cascade of the linear prediction 6-pole spectrum synthesis and the 3-tap pitch synthesis filters. The output of the cascade is deemphasized using a fixed single-pole filter, D/A converted, highpass filtered at 100 Hz, and lowpass filtered at 3.2 kHz to produce the analog output speech.

2.2 Algorithm Modifications

The pitch prediction method used in the APC-NS algorithm requires the computation of an autocorrelation of a 256-point window for lags up to 134. In our MAP-300 implementation of the APC-NS coder, this autocorrelation was computed using two 256-point FFT's. Since the TMS32010 processor is not well-suited for the computation of large FFT's, we decided to compute the autocorrelation directly in the time domain.

Preliminary estimates of the computational requirements indicated that performing a straightforward autocorrelation over the pitch window would be prohibitively expensive in TMS32010 cycles. We therefore modified the original algorithm to reduce the number of computations. Among the various modifications we investigated were the following:

1. downsampling of the preemphasized waveform by factors of 3, 4, and 5, before performing the autocorrelation, both with and without filtering;
2. a modified autocorrelation in which, for each autocorrelation lag, only every Nth product contributed to the sum (for N of 3, 4, and 5); and
3. averaging 3, 4, or 5 samples together before finding the autocorrelation (i.e., a very simple lowpass filter and downsample, as in (1)).

We tested these methods to find the one that produced the same autocorrelation peak as the original most often. We determined that using the modified autocorrelation, reducing the summation terms by a factor of 4 to find the approximate autocorrelation peak, and computing an exact autocorrelation over only 9 lags around that peak, produced the best compromise between accuracy (as determined by the original algorithm) and computational complexity. This modification resulted in a reduction of approximately 60% in the number of computations required for calculating the autocorrelation function.

2.3 Algorithm Implementation and Verification

The implementation of the APC-NS algorithm for the VAD system proceeded in two stages: implementation of a 16-bit fixed-point version of the algorithm and TMS32010 assembly language implementation of the fixed-point algorithm. The implementation was verified as it proceeded through each stage, and again when each stage was completed.

The fixed-point version was based on our Fortran floating-point implementation. Successive modules of the floating-point version were replaced by fixed-point modules as the implementation progressed. To verify the correctness of each replacement, the Fortran code was modified to include a fixed-point version of the module under development. The inputs to the module were converted to fixed-point and the outputs from the module were converted back to floating-point. The latter conversion usually involved scaling the fixed-point outputs, as many of the fixed-point modules employed "block floating point" techniques. The fixed-point and floating-point versions of the module were run in parallel, in the context of the complete algorithm, on complete utterances. The outputs of the two modules were compared, and each difference that exceeded the tolerance for that particular module was reported to the user. This procedure allowed us to compare the fixed-point and floating-point modules over a substantial amount of data. The tolerances varied across the modules. For example, preemphasis, which is a relatively simple calculation, could be expected to be less affected by fixed-point considerations than the lowpass recursive filter, and so the tolerances were small. In some cases, the comparison itself had to be modified. Instead of directly comparing autocorrelation coefficients, we compared these coefficients normalized by the corresponding R0. This approach cancelled out scaling effects.

When the fixed-point version was debugged, we removed the floating-point version of the module and ran the system over the database, comparing segmental and long-term signal-to-noise ratios (SNR) obtained to those from the previous generation of the system (containing the floating-point version of the module in question). Our overall goal was to keep the SNRs of the fixed-point system to within 0.5 dB of the floating-point SNRs. The modules caused generally very small decreases in the SNRs, on the order of a few hundredths of a dB. For some modules, the SNR actually

increased for the fixed-point version. We also used informal listening tests to ensure that the fixed-point version was not introducing any artifacts into the output speech. Table 2-1 compares segmental and long-term SNR results for the original floating-point system and the fixed-point system. SNR values are given for both the clear channel and the 1% bit-error-rate (BER) channel, for each of six sentences and as averaged over all six sentences, the six sentences represent three male and three female speakers, with each speaker saying a different sentence.

The fixed-point implementation used a fixed-point library of basic operations, such as ADD, MULTIPLY, SHIFT, and DIVIDE. We created this library to mimic the functionality and arithmetic precision available on the TMS 32010. Using this library in the fixed-point implementation of the algorithms provided a reliable guide for the assembly-language implementation.

The TMS32010 assembly-language version was implemented module-by-module, tracking the development of the fixed-point implementation. The individual modules were debugged using a TMS32010 simulator running on the VAX. Once debugged, a module was tested using actual speech data created by the fixed-point implementation, and the outputs were compared with the outputs created by the corresponding fixed-point module. Exact equality was required at this point, and was easily verified using file comparison utilities. As each module was verified, it was combined with the previously-developed modules, and the combination was again checked for exact equality with the fixed-point implementation. Finally, the entire database was processed by the transmitter assembly-language program (including the bit-streaming and error protection modules); the results of this were then processed by the receiver program, and the results were compared to the synthetic speech produced by the fixed-point implementation. Again, exact equality was required.

This procedure tested the TMS32010 assembly code quite thoroughly. During hardware/software integration, the only problems we encountered were discrepancies between the TMS32010 simulator and the actual performance of the chip. Because these were systematic bugs, they were relatively easy to find and fix. In fixing and verifying the corrected operation of the code executing in the chip, we again performed comparisons of intermediate results.

	Clear Channel		1% BER Channel	
Sentence	Floating	Fixed	Floating	Fixed
BV1M	12.501	12.507	3.861	4.234
	(13.489)	(13.664)	(9.257)	(9.244)
JE1M	11.065	10.817	6.288	6.616
	(12.095)	(12.205)	(8.380)	(8.054)
JS1F	15.753	16.080	5.314	4.574
	(15.208)	(14.570)	(8.604)	(8.103)
LS1F	14.481	14.235	-.776	-1.587
	(13.425)	(13.364)	(6.238)	(7.317)
MP1F	15.572	15.814	6.273	5.601
	(15.649)	(16.889)	(8.516)	(9.810)
RH1M	12.601	12.285	6.046	4.949
	(12.735)	(10.895)	(7.460)	(6.776)
Average	13.662	13.623	4.501	4.065
	(13.767)	(13.598)	(8.076)	(8.217)

Table 2-1: Long-term and segmental signal-to-noise ratio comparison for floating and fixed point APC-NS simulations. Segmental SNRs are given within parentheses.

The integration of the assembly-language modules was aided by a package of memory allocation macros we developed. These macros provide symbolic definitions of variables and buffers, allowing locations to be allocated at assembly time. This facility meant that we did not have to define and maintain a memory map during the development process.

The 68000 control program was developed using the XMD debugging package for the 68000. This package permits breakpointing and memory and register examination. We verified the operation of the 68000 code by creating transmitter and receiver "stubs". The stubs are simple TMS32010 programs that move data from input to output buffers, and manipulate the corresponding buffer flags appropriately. The stubs decimated and interpolated the data to match the sampling rate and the bit rate, enabling us to verify the continuity of the signal across frame boundaries.

3. 2.4 KBIT/S HDV CODER

3.1 Algorithm Description

A detailed description of the 2.4 kbit/s HDV coder algorithm is given in [4]. The optimized HDV coder algorithm may be summarized as follows. In the transmitter, the analog speech is lowpass filtered at 5 kHz, sampled at 10 kHz, and divided first into frames of 20 ms duration and then into 9-frame blocks. A variable frame rate (VFR) algorithm is used to select and transmit only 6 frames of data every block, along with a block header to identify the transmitted frames. For every frame selected by the VFR algorithm, the following quantities are transmitted: a synchronization bit, voicing status, pitch, speech signal energy, 12 linear predictor coefficients (log area ratios), and 3 selected spectral deviations between the log spectrum of the speech signal in the frame and the log spectrum of the all-pole model. These quantities are quantized, coded, partially error-protected, and transmitted across the channel.

At the receiver, the data for the untransmitted frames are regenerated by linear interpolation between adjacent transmitted frames. The output speech of the coder is synthesized, pitch-synchronously for voiced frames and every 10 ms for unvoiced frames, by generating the excitation signal using the spectral deviations and applying it to the all-pole synthesis filter. The digital output is D/A converted and lowpass filtered at 5 kHz to produce the analog output speech.

3.2 Algorithm Modifications

We initially reviewed in detail the HDV coder algorithm and identified several parts of the algorithm for potential simplification of the computational complexity. Each simplification was implemented as a user-specified option in the coder program. To evaluate the algorithm modifications, syntheses obtained using the options were compared by informal listening tests to the original HDV coder output. For all tests we used our phoneme-specific database of 6 utterances (see Section 3.3.1 of [4]). Our effort to simplify the HDV coder algorithm is described in the following subsections. We note that the HDV coder that incorporates all the simplifications recommended below produced about the same overall speech quality as did the original HDV coder.

3.2.1 Smoothing of the Residual Signal Spectrum

The cepstral smoothing of the residual signal spectrum (which requires two 256-point FFT's) could be replaced by the computationally simple filter-bank smoothing method (see Section 4.3 of [4]). We found no perceivable change in speech quality or intelligibility as a result of this replacement.

We then simplified the coder by revising the filter-bank smoothing method itself. In the original scheme, a power spectrum of the residual signal is computed with a 256-point FFT. Each deviation is obtained by averaging (in dB) the spectral components that occur in a band centered on the harmonic location. The averaging band has width equal to the pitch frequency. The mean of all deviations in the 5 kHz band is then evaluated and subtracted from each deviation. In this scheme, the number of spectral components used to compute each deviation is dependent on the pitch frequency. We modified the smoothing algorithm so that each deviation is computed as an average of five equally spaced DFT spectral components in the same band as in the original algorithm. Also, because of our decision to transmit only the first three deviations (see below), the high frequency components of the spectrum were no longer required. With the new approach, unnecessary high frequency spectral components are not evaluated and the number of computations each frame is fixed. Since only three deviations are evaluated each frame, we cannot compute the mean of all the deviations as in the original algorithm. To normalize the deviations, however, we subtract the energy (in dB) of the residual signal from each deviation, since the residual signal energy is, in fact, approximately equal to the mean of all deviations in the 5 kHz band. We compute the residual signal energy directly from the zeroth autocorrelation coefficient of the speech signal and the normalized error, which are evaluated during LPC analysis.

Our modified smoothing algorithm caused some changes in the distribution of the deviations. We computed the statistics (Table 3-1) of the new deviations for the purpose of revising the quantization ranges. For each deviation we obtained the mean, median, and 5- and 95- percentile points from a database of 24 sentences; the six phoneme-specific utterances and 18 utterances (9 male and 9 female) from a previous contract effort (the pitch and voicing study [5]). Our quantization routine was updated with these 5- and 95- percentile values. Quantization of the deviations did not produce any perceivable change in the HDV coder output.

Informal listening tests indicated that our modified smoothing technique resulted in a slight improvement in speech quality for male speech. The synthesized male speech was less "bassy" and sounded slightly more natural. No perceivable changes were noted in the intelligibility or quality of the utterances from the female talkers.

Deviation #	Mean	Median	Percentile 5%	Percentile 95%
1	−1.295	−0.775	−7.329	2.754
2	−1.756	−1.338	−8.287	2.927
3	−3.106	−2.501	−10.343	2.054

Table 3−1: Statistics of the first three deviations.

3.2.2 Extraction of the Harmonic Deviations

The second simplification dealt with the way the 3 harmonic deviations were computed. The original algorithm computes all deviations over the 5−kHz frequency band and transmits the first two harmonic deviations and a third one around a formant peak beyond the second harmonic. We examined the transmission of the first three deviations rather than the first two and a third around a formant peak. To implement this change, the computation of the LPC spectrum in the transmitter and receiver for evaluation of the formant peaks was bypassed and the first three deviations were always transmitted. Informal listening tests indicated that this change did not degrade the speech quality or intelligibility of the HDV coder output. The foregoing two simplifications reduce considerably the computational complexity of the coder by eliminating four 256−point FFT computations per frame: two for LPC spectrum computations and two for cepstral smoothing. Also, the deviations can be computed over a smaller frequency band with a low−order DFT.

3.2.3 Receiver Interpolation

The receiver of the original HDV coder uses the so-called spectral interpolation to regenerate the harmonic deviations for the untransmitted frames [recall that the HDV coder uses variable frame rate (VFR) transmission] and to determine the harmonic deviations for pitch-synchronous updating of the synthesizer excitation. This interpolation required three 256-point FFT's for computing the spectrum of the LPC all-pole filter for the two frames involved in the interpolation and for the case corresponding to the interpolated LPC parameters. In contrast, the direct interpolation method interpolates the deviations in dB linearly. We first tested the direct interpolation method. Informal listening tests indicated that direct interpolation caused roughness in the synthesized speech. This result suggested that spectral interpolation might be required for good speech quality. We reexamined in detail the interpolation algorithm with the intent of simplifying the spectral interpolation method or possibly modifying the direct method. The results of our experiments are given below.

In the original HDV coder interpolation routine, the harmonic spacing used for all computations of the half-frame interpolation is the present frame pitch frequency. This is a reasonable approach since the pitch is not interpolated within a frame. However, since the pitch of the next frame may differ somewhat from the current frame value, we modified the routine to compute the next frame LPC spectral components at the harmonic spacing equal to the next frame pitch frequency. Interpolation is then done between the spectral components at the harmonics of the two frames without regard to the specific frequency locations of the harmonics. Although this modification did not reduce the computational complexity, it did produce an improvement in speech quality. Synthesis with this modification had more clarity and naturalness as compared to the original HDV coder output.

We also modified the spectral interpolation method used for regenerating deviations for the untransmitted frames. In the original program, the harmonic spacing for each frame is determined by the transmitted or interpolated pitch frequency of the frame. However, the LPC spectral components are computed by 256-point FFT's rather than by DFT's and interpolated in frequency to evaluate the interpolated deviations. We modified this routine to compute the LPC spectral

components with DFT's at the appropriate harmonics for each frame. Interpolation is then done between the components at the harmonics as in the new half-frame interpolation method described above. Informal listening tests indicated that this modification did not degrade the speech quality of the HDV coder output. The foregoing modifications to the interpolation routines imply that if the first three deviations are transmitted, we need to compute only three LPC spectral components at each frame to determine the interpolated deviations, with the deviations at all other harmonics set to zero. (In the original program, interpolated deviations are computed at all harmonics in the 5 kHz band.)

Using the modified smoothing algorithm (see Section 3.2.1) to extract deviations, we synthesized speech with direct interpolation, simplified spectral interpolation, and the original spectral interpolation method. Informal listening tests indicated that synthesis with the original spectral interpolation method was slightly more natural than synthesis with the simplified technique. However, we found that synthesis with direct interpolation was quite similar to synthesis using the original spectral interpolation. For several of the male speaker utterances, direct interpolation was preferred over the original spectral interpolation method. From the results presented above, we decided to use the modified smoothing algorithm with direct interpolation of deviations for untransmitted frames and at half-frame intervals.

3.2.4 Excitation Signal

We simplified the computation of the excitation signal at the receiver, as described below. In the original HDV coder, the excitation signal is obtained by an inverse DFT of a complex spectrum whose phase is zero for voiced frames and random for unvoiced frames and whose amplitude is constant at all frequencies except the first three harmonics where we apply the transmitted deviations. Two inverse DFT's, each one pitch period long (10 ms for unvoiced frames), are computed every frame. To simplify this computation, we expressed the excitation spectrum as two additive components, one that represents the spectrum without harmonic deviation corrections and a second that contains the changes in the spectrum caused by the deviations. The first component can be defined in the time domain directly (without an inverse DFT) as the standard binary pulse/noise excitation signal. The time-domain signal of

the second spectral component can be computed efficiently with an inverse DFT involving only three non-zero frequency components since the excitation spectrum is modified by the deviations at only three frequencies. The two time-domain signals are added to produce the desired excitation signal.

For voiced frames, the above method was easily implemented. Tests indicated that the simplifications did not cause any changes in the synthesized speech. For unvoiced frames, we tested the following method for construction of the excitation signal. Spectral components of a uniformly distributed random noise signal were computed at 100, 200, and 300 Hz: the locations of the transmitted deviations. Changes in these spectral components caused by the deviations were obtained and a signal representing these changes was computed by inverse DFT. This signal was added to the random noise to form the total excitation signal. Synthesis with the above method was rough and therefore not as natural as the original HDV coder output. We also tested the above method and the original method with all deviations set to zero and obtained similar results (i.e., synthesis with the simplified method was more rough and less natural). This test indicated that the time-domain generated random noise signal was the cause for the roughness in the speech. In the original HDV coder the unvoiced excitation without spectral corrections is gaussian distributed. We repeated the test using gaussian rather than uniformly distributed random noise and found a slight reduction in the roughness; however, the resulting speech was still inferior to the original HDV coder output. To examine further the apparent enhancement in speech quality caused by a DFT generated excitation, we added an option to our LPC vocoder to compute the unvoiced excitation signal (via inverse DFT) from a flat spectrum with random phase and found a perceivable improvement in the naturalness of the synthesized speech. From these tests we concluded that the unvoiced excitation signal should be computed from an inverse DFT of a random-phase, constant amplitude spectrum. To avoid DFT computations each frame, we precomputed four excitation signals with the DFT method and chose randomly one of them for the computation of each unvoiced excitation signal with spectral corrections. The roughness in the synthesis was eliminated. However, unvoiced sections were somewhat buzzy because of repetition of the four signals. With ten precomputed excitation signals we were able to eliminate the buzziness and obtain synthesis that was approximately equivalent to the original HDV coder output. The technique requires the storage of 1000 excitation samples and phase information for the

computation of the signal with spectral corrections (30 phase values, 3 for each of the 10 signals). Should storage be a problem, the excitation signal may be computed using DFT. The DFT can be simplified to a summation of cosines since the spectral amplitude is constant.

3.2.5 Variable Frame Rate Transmission

We reviewed the variable frame rate scheme with the intent of simplifying the implementation of the algorithm and distributing the computational effort over the nine-frame block. In the original algorithm, all analysis is postponed until the full block of data is available and then each of the 56 possible transmission sequences is independently evaluated. Recall that the last frame in the block is always transmitted so that 5 out of 8 frames must be chosen for transmission. This approach was appropriate for our FORTRAN simulation of the HDV coder since it was easy to implement and allowed us to experiment with various block sizes and error measures. However, for real-time implementation, the above approach is not efficient. Because of the independent treatment of each transmission sequence, many of the error measures are unnecessarily recomputed. For example, 35 of the 56 possible transmission sequences contain the transmission of the first frame of data in the block. Quantization error for these parameters is computed each time one of the 35 transmissions sequences is examined. The quantization error could be computed once and stored for use in the evaluation of each of the sequences. Also, note that these errors could be computed as soon as the first frame of transmitted parameter data is available. Similarly, when the second frame of data is available, we could compute the quantization error for transmission of the second frame and also the interpolation error that would occur if the first frame was not transmitted. With this approach, redundant calculations are eliminated and the computational effort can be distributed over the 9 frames. All 56 possible transmission sequences can be tested with a maximum computation each block of 9 quantization errors and 21 interpolation errors for each parameter.

To simplify the VFR implementation, we separated the VFR analysis into two parts: (1) frame-by-frame computation of quantization and interpolation errors for the transmitted parameters and (2) selection of the optimal transmission sequence after

each block of 9 frames has been analyzed. The first part of the analysis computes and stores each frame a maximum of four errors for each of three parameters: pitch, gain, and LPC coefficients. The four errors are the quantization error for the current frame and the interpolation error that would occur if 1, 2, or 3 of the previous frames were not transmitted. Note that in the original VFR scheme, the LPC coefficient error is weighted by the ratio of the current frame energy to the maximum frame energy in the block. To incorporate the energy weighting in the simplified scheme, at each frame the LPC coefficient error is weighted by the current frame energy and a value for the maximum frame energy in the block is updated. In the second part of the analysis all LPC coefficient errors are normalized by the maximum frame energy.

The second part of the VFR analysis is carried out, as mentioned above, after all frames in the block have been analyzed. At that time, mandatory frame transmissions because of voicing transitions are identified, appropriate errors for each permissible transmission sequence are combined, and the optimal transmission sequence is chosen.

The data storage requirements for the new VFR implementation are essentially equivalent to those of the original scheme. To perform the new frame-by-frame error analysis, only 5 frames of parameters need to be stored. The original VFR scheme requires the storage of 10 frames of parameter data (9 frames of the current block and the last frame of the previous block). However, the new implementation requires the storage of error data for each of the parameters, which approximately offsets the above savings. The main advantages of the VFR modifications are the elimination of redundant computations and the distribution of the computational complexity over the 9 frames of the block, as noted above.

3.2.6 Pitch and Voicing Algorithm Simplification

One of the simplification ideas given in our original technical proposal was to simplify the pitch and voicing algorithm. What we had in mind was to replace the AMDF algorithm used in the HDV coder by a simpler algorithm such as the Gold pitch detector. As part of our recent pitch and voicing study project [5] we found, however, that the Gold algorithm is significantly worse than the AMDF algorithm. Therefore, we decided to retain the use of the AMDF algorithm in the HDV coder.

3.2.7 Effect of 100-Hz Highpass Filtering of Input Speech

We examined if using a 100-Hz highpass filter on the input speech required any algorithm change (re-optimization) as to which harmonic deviations should be transmitted. The six sentences of our database were filtered with an existing 257-point FIR Hanning window filter with a highpass cutoff of 200 Hz. This filter caused severe attenuation to the low frequency components (-44 dB at 140 Hz). We used this filter to examine the worst case effect highpass filtering would have on the HDV coder output. We processed the highpass filtered speech through our LPC and original HDV coders. We did not quantize the deviations since the highpass filtered input speech caused a substantial change in the distribution of the first deviation. Informal listening tests indicated that the HDV coder output was still more natural sounding and less buzzy than the LPC coder output. However, the HDV coder speech was quite rough and reverberant. Since highpass filtering should primarily affect the first deviation, we also tested the transmission of the second, third, and fourth deviations. Much of the rough and reverberant characteristics were removed from the output speech, but the output speech sounded less natural; this indicates that the transmission of the deviation at the first harmonic is essential for high quality speech. Highpass filtering of the HDV coder output with the first three deviations transmitted was found to remove many of the distortions while retaining the natural characteristics of the speech. From these tests we concluded that the deviation at the first harmonic must be transmitted to ensure high quality synthesized speech and, if the input speech is highpass filtered, the synthesized output should also be highpass filtered.

4. REAL-TIME HARDWARE AND SOFTWARE DESIGN AND IMPLEMENTATION

4.1 Hardware Design

In this section, we discuss the hardware functionality of the Voice A D (VAD) system. The VAD hardware is documented in detail in the report [6]. The goal of the VAD hardware design was to produce a general-purpose speech and signal processing device capable of supporting the real-time implementation of BBN's 16 kbit/s APC-NS high-quality voice coding algorithm in full-duplex.

The VAD printed circuit board measures 12" x 18", and uses about 30 Watts total power. It is packaged as a portable unit that can communicate directly, or through a synchronous modem, with another VAD unit.

A block diagram of the VAD architecture is shown in Figure 4-1. The VAD system has a highly flexible 68000-based architecture. The computational power of the system is provided by three TMS32010-based signal processing modules. A high quality analog I/O interface, 12-bit A/D/A conversion, four programmable serial I/O ports, and a dedicated serial bitstreaming port provide the input and output paths between the VAD system and the external environment.

4.1.1 68000 Circuitry

The primary function of the 68000 is to control the system data flow. The TMS32010 memories and the analog and digital I/O ports are mapped into the 68000 memory space, allowing the 68000 to communicate with all of the devices on the board. Interprocessor communication among the TMS32010 modules and between the I/O processes and the computational processes is implemented by the 68000. The analog and digital I/O processes are performed asynchronously, and they communicate with the 68000 via interrupts. The 68000 allows seven levels of interrupts. The priorities of the TMS32010 modules and the I/O devices are hardware programmable.

VAD SYSTEM BLOCK DIAGRAM

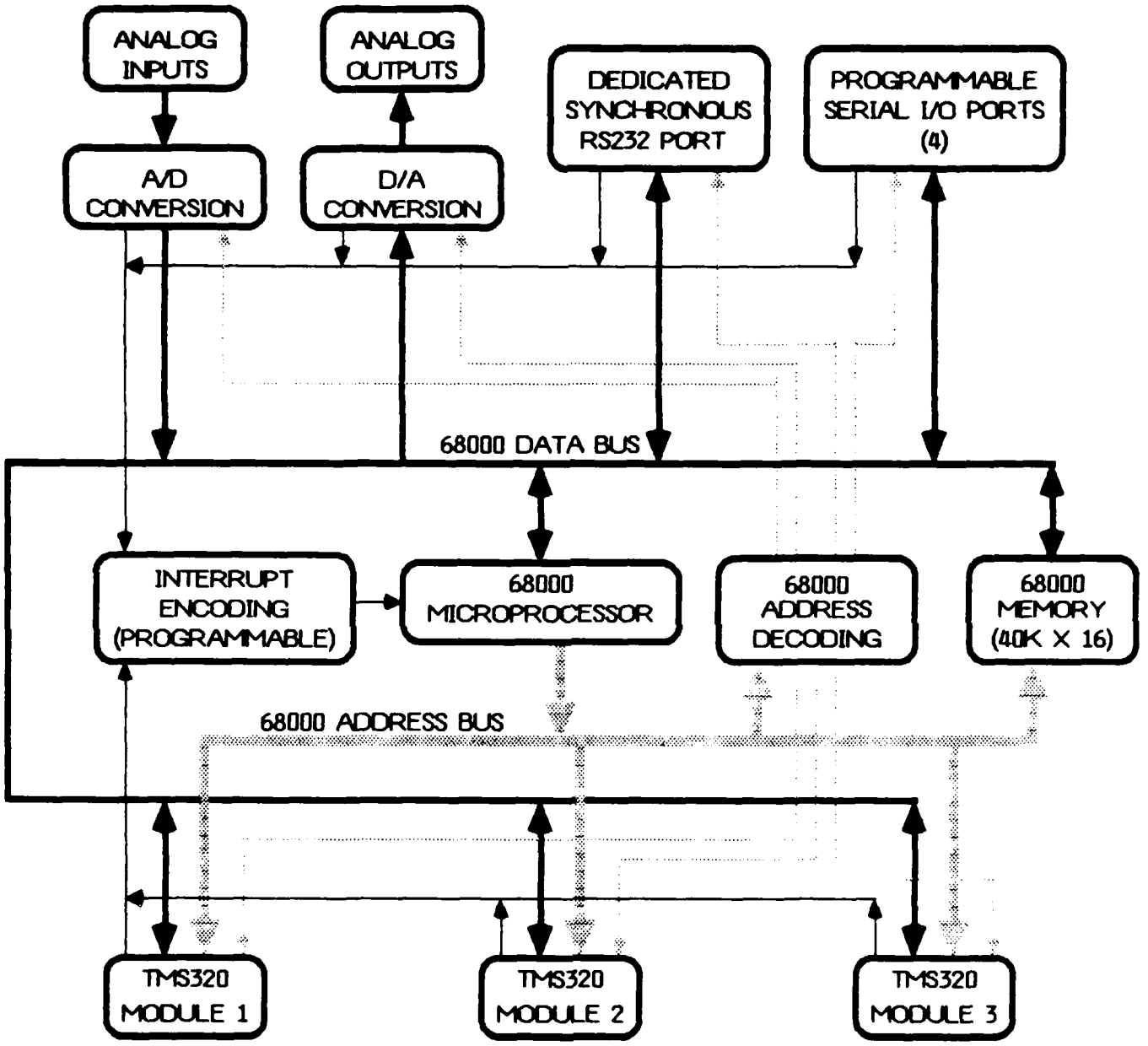


Figure 4–1: Hardware architecture of the voice A/D system.

The 68000 system includes 40K words of memory configured as 32K words of EPROM and 8K words of RAM. In addition to the 68000 program, the EPROM contains the TMS32010 programs. These programs are downloaded into the TMS32010 program memory space upon startup.

4.1.2 TMS32010 Module Description

Figure 4-2 is a block diagram of a TMS320 module of the VAD system. Each module consists of a TMS32010 microprocessor, its program memory, external data memory, and address decoding circuitry. The TMS32010 is a VLSI NMOS processor with a 200-nanosecond instruction cycle, 16 bit I/O and data paths, 16 x 16 single-cycle multiply and 32-bit ALU and accumulator designed for digital signal processing applications. It has 144 words of on-chip data RAM and can access up to 4K words of external program memory.

The TMS32010's limited internal data RAM necessitates the addition of external data memory. Dual-ported RAM, addressable by the TMS32010 and the 68000, is used to implement this memory. The dual-ported RAM is constructed by interleaving memory accesses by the TMS32010 and the 68000 in a single TMS32010 cycle. Interprocessor communication between the 68000 and the TMS32010 can be realized via signaling flags in this memory space. The processors are also able to signal each other via hardware interrupts.

A similar dual-ported RAM system is used for the TMS32010 program memory. The TMS32010 programs are stored in 68000 EPROM and downloaded to the TMS32010 program memory when the VAD system is powered on. This allows the TMS32010's to execute from their program memories at full speed. In addition, it allows the 68000 access to all of the TMS32010 memory, creating a flexible architecture that lends itself to a variety of other applications.

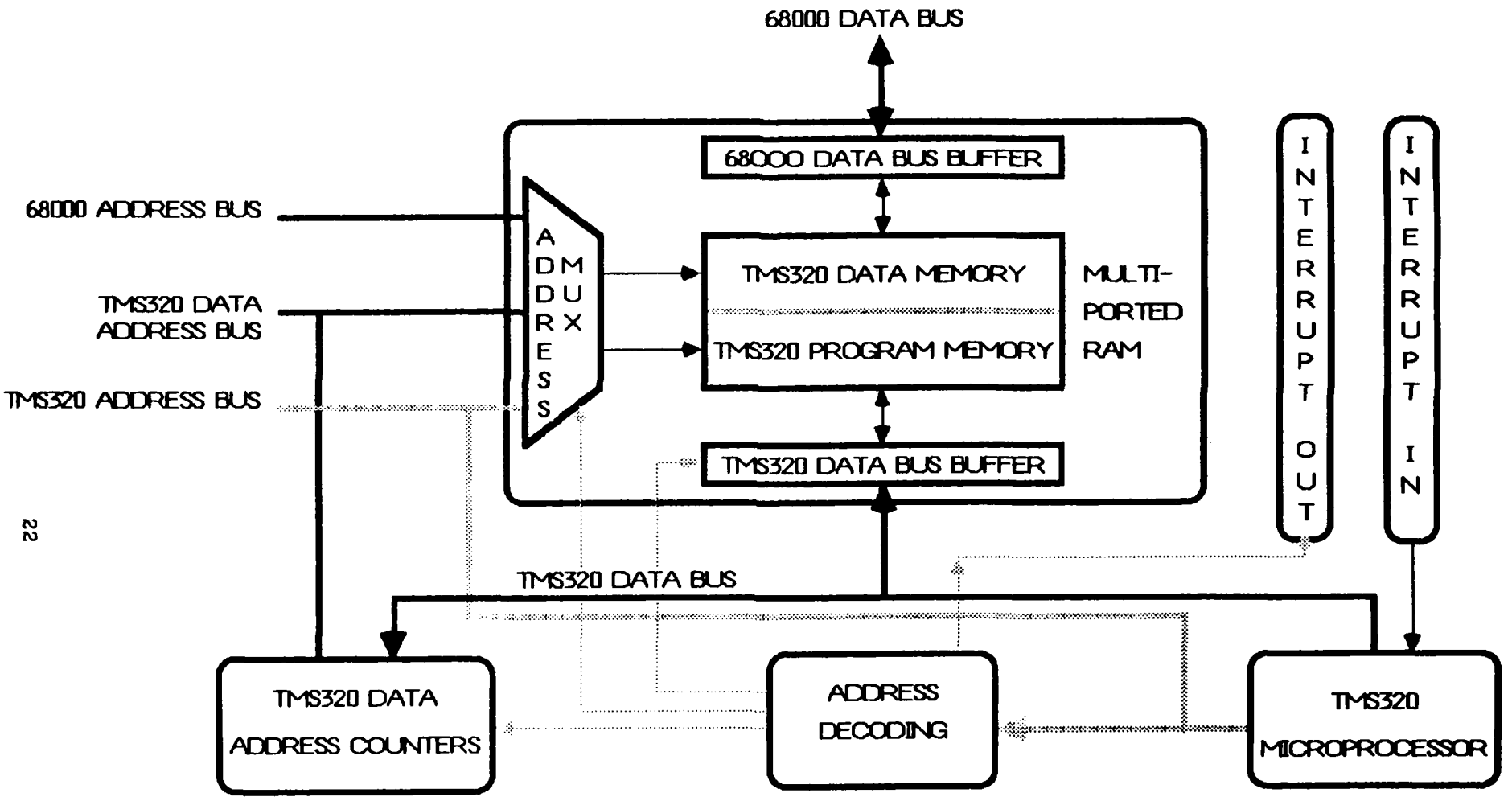


Figure 4-2: TMS320 module of the voice A/D system.

4.1.3 Analog and Digital I/O

The VAD system has eight I/O channels. Three of these channels are analog and the remaining five are digital. The three analog input and output pairs are a tape recorder interface, a telephone interface, and a handset interface for a high quality dynamic microphone. The tape recorder interface has balanced differential 600 ohm inputs and outputs, with signal levels of 0 dBm The microphone has a bandpass frequency response with 3 dB points at 100Hz and 4700 Hz. The telephone interface includes a DTMF decoder and on/off hook detection.

The analog signals are converted to and from digital data using 12-bit linear A/D and D/A converters whose sampling rates are independent and programmable. The A/D and D/A interfaces include 16-word FIFO buffering to reduce the load on the 68000.

A variety of digital I/O paths are available on the VAD board. VAD systems communicate via a dedicated RS232 synchronous modem interface with a programmable clock rate. The modem clock is phase locked with the sampling clock to avoid clock drift. In addition to the dedicated synchronous channel, four programmable asynchronous/synchronous RS232 ports supporting a variety of protocols are available on the board.

4.2 Software Design

The VAD APC-NS software is documented in detail in the report [6]. The software system architecture is based on the hardware architecture described in the previous sections. This software architecture defines the segmentation of the system software into software modules, and the distribution of these modules within the multi-processor hardware system. It also addresses the issues of inter-module communication and module invocation control. In developing this design, the goals of graceful system degradation and straightforward system testing and debugging were considered.

The system software is divided into three basic types of software modules:

- Interrupt Routines
- TMS Programs
- Data Distribution Process.

Interrupt Routines run in the 68000, and transfer "external world" data (A/D samples in, coded bits out, coded bits in, D/A samples out) between the external world devices and the double-buffers that reside in the 68000 memory. There is one interrupt routine to handle each of these four types of data, and each interrupt routine transfers data to/from one of four buffers. Each half of each buffer has a full/empty flag associated with it. When an interrupt routine fills (or empties) half of a buffer, it sets (or clears) the appropriate full/empty flag and goes on to the other half of the same buffer. The other half of the buffer may not be available, as during system start-up, or if parts of the system are not keeping up with real-time. In this case, the interrupt routine "marks time" by either discarding input data or producing "fake" output data (for example, it repeats the most recently output speech sample).

The second type of software module is *TMS Programs*. There is one TMS program for each of the two TMS32010 processors used in the APC-NS system; these programs perform the computations necessary to transform the input speech samples into the output bitstream ("analysis"), and the input bitstream into output speech samples ("synthesis"). These tasks are divided such that the entire analysis TMS program runs in one of the TMS processors, and the synthesis TMS program runs in the other processor. Input and output data for each TMS program is stored in buffers in dual-ported memory (accessible to both the TMS processor running that program and to the 68000), and each of these buffers has a full/empty flag associated with it. A TMS program waits until its input buffer(s) are marked full, and then performs its processing, marking each input buffer empty when it no longer needs the data contained in the buffer. When the TMS program has completed its processing, it waits until its output buffers are marked empty. It then fills them and waits until a new frame of input data is available for processing.

The third type of software module is the *Data Distribution Process*. This module runs in the 68000. It transfers data between the buffers accessed by the interrupt routines and the buffers accessed by the TMS programs. It can also transfer data between buffers accessed by two or more different TMS's (although there are no such

buffers required for the APC-NS system). It includes a table of source and destination buffers, and transfers data from source to destination when the source buffer is marked full and the destination buffer is marked empty. The data distribution process simply loops through all the source/destination buffer pairs in its table, and performs data transfers whenever it can. For the Bitstream-In data, the data distribution process also performs frame-synchronization acquisition and maintenance.

We note that in the software system design outlined above, only the interrupt routines are invoked directly by real-time events. The remainder of the modules in the system are data-driven: they execute whenever their input data is available and their output data buffers are empty. This means that the system (except for the interrupt routines) operates using the same control structure whether it is being run in real-time or tested/debugged in non-real-time.

A significant result of this data-driven system design is that the overall system delay is minimized: data can be processed as soon as it is available rather than at frame-boundaries only. We have measured the overall system delay (not including channel transmission delay) introduced by our APC-NS real-time implementation to be about 115 ms.

4.3 System Performance

As indicated above, the VAD APC-NS speech coder system introduces a total delay of about 115 ms, between voice input and synthesized voice output (not including any delays in transmission). This results from one (33.75 ms) frame to accumulate a buffer of input samples, one frame to analyze and begin to transmit data bits, one frame to accumulate a buffer of received data bits, 5.99 ms to synthesize a buffer of output samples, 4.05 ms of additional delay before these samples are output, and about 2 ms of delay for buffer manipulation in the MC68000.

We measured the computational load placed on the VAD hardware by executing the real-time APC-NS coder. We measured the execution time of the MC68000 code for interrupt and data distribution processing by causing each routine to set a bit in the

MC68000 hardware debug register upon entry and to clear the bit before exiting. We used a Tektronix 1240 Logic Analyzer to examine the state of the debug register in real-time. During normal system full-duplex operation (exclusive of the sync acquisition and maintenance routines) the MC68000 programs require about 14.04 ms per 33.75 ms frame, or about 41% of real-time. Of this, about 1.9 ms is used by the data distribution process, and 12.14 ms is used for interrupt processing.

We used the TMS simulator to measure the single-frame execution time of the entire APC-NS transmitter to be about 27.28 ms, or about 80.8% of the available 33.75 ms frame time. The entire APC-NS receiver requires about 5.99 ms, or about 17.7% of the frame time.

The MC68000 program uses about 4K bytes for program storage and about 30K bytes for constant and variable data storage (including the storage of the TMS transmitter and receiver program code, which is downloaded by the MC68000 from its data memory into the TMS program memories).

The TMS transmitter program occupies all of the available 4096 16-bit words of program memory; the receiver program requires about half this amount of program memory. The transmitter uses 3800 16-bit words of external data memory, and the receiver uses approximately 1800.

We have demonstrated that the VAD APC-NS coder produces toll quality speech with no channel errors. We have used the government-provided channel error simulation box to introduce 1% channel errors; under this condition the coder produces high quality speech. In fact, the effect of 1% bit-errors is just barely audible. In addition, the coder performs satisfactorily in tandem with LPC-10.

5. COST STUDY FOR POTENTIAL PRODUCTION

In this project, we designed and implemented a real-time Voice A/D system for transmission of toll-quality speech at 16 kbits/s. Three VAD units were produced and delivered to the Government. Also, we studied the cost implications of volume production of the system. The costs were estimated assuming the use of BBN Communications Corporation (BBNCC) manufacturing personnel and, therefore, include labor, material handling, general and administrative, and cost of capital estimates based on BBNCC current rates. Material cost estimates are based on current quotes from distributors. The labor estimates include a cost-effective labor mix that utilizes both BBNCC U.S. and Hong Kong manufacturing facilities.

Costs are estimated for production of 100, 500, 1000, and 10000 units. The estimates are divided into an initial cost of introduction (independent of number of units), and then a per unit production cost.

The current VAD system implementation is based on integrated circuit technology that was available (or soon-to-be available) at the time of the VAD system design. The signal processing functions of the VAD are implemented using Texas Instruments (TI) TMS32010 digital signal processors, as discussed in detail in Section 4. The next generation TI digital signal processor, the TMS32020, is now available; an enhanced CMOS version, TMS320C25, may be available in the third quarter of 1986. Since implementation of the system utilizing new (i.e., current state-of-the-art) technology components would result in cost savings, as well as savings in size and power consumption, it is instructive to estimate production costs for the VAD system redesigned with that newer technology. A preliminary design of such an implementation is given in this section. Production cost estimates are included for the redesigned system.

Another alternative for implementation of the speech coding units is the use of custom or semi-custom LSI/VLSI components. This alternative is also discussed in this section.

5.1 Cost Estimates for Current VAD System Implementation

Cost estimates for production of VAD system units are given in Tables 5-1 and 5-2. Table 5-1 details the cost per unit for various production quantities. Table 5-2 gives the cost to introduce, defined as the nonrecurring cost for initial set-up of automated test equipment, manufacturing aids, etc Note that these estimates are for the processing board only, and do not include additional packaging or switches, connectors, etc., that are not on the printed circuit board.

As seen in the tables, the cost per unit in quantities of 100 are approximately $2300 per unit. Cost to introduce is about $30,000, or $300 per unit. Thus, total cost for production of 100 units is approximately $260K (or $2600 per unit).

Production costs decrease approximately 12% to $2026 per unit in quantities of 10,000. Thus, total cost for production of 10,000 units would be $20.3 million (or $2029 per unit).

5.2 VAD System Implementation Utilizing State-of-the-Art Technology

Below, we describe a preliminary design of the VAD system using technology that has been announced subsequent to the design of the current VAD system implementation. Production cost estimates follow that description.

5.2.1 VAD System Redesign Utilizing New Technology Components

The redesign of the VAD system is based on the TI TMS32020 digital signal processor. Two TMS32020 processors and associated circuitry are incorporated onto a single 5.6" x 6.0" printed circuit board, using less than 15 Watts total power. The board is designed to function as a single full-duplex coder, two transmitters, or two receivers. Provision is made for upgrade to the TMS320C25 processor, thus allowing for two full-duplex coders on the single board. Cost estimates given here, however, are for the TMS32020 version only. Parts costs for the upgrade have not been estimated, but are expected to be similar to the cost for the TMS32020 design.

Product Costing	QUANTITIES			
	100	500	1000	10000
Base Material Dollars	1348.67	1304.41	1260.78	1194.28
Mat'l Handling at 24.4%	329.0755	318.2760	307.6303	291.4043
TOTAL MATERIAL	1677.745	1622.686	1568.410	1485.684
Labor Hours				
HK hours	11.3	11.3	10.7	9.6
$11.33/hr	128.029	128.029	121.231	108.768
US hours	4.5	4.5	4.3	3.8
$45.51/hr	204.795	204.795	195.693	172.938
TOTAL LABOR	332.824	332.824	316.924	281.706
SUB TOTAL	2010.569	1955.510	1885.334	1767.390
CCMD General & Admin. at 13.7%	275.4480	267.9049	258.2908	242.1325
Cost of Money				
US at $1.67986/hr	7.55937	7.55937	7.223398	6.383468
HK at $0.17048/hr	1.926424	1.926424	1.824136	1.636608
CCMD G & A at 0.00496/$	9.972425	9.699330	9.351258	8.766256
TOTAL PRODUCT COST	2305.476	2242.600	2162.024	2026.309

Table 5-1: Per unit production cost estimates for the VAD system as currently implemented

	HRS	RATE($)	COST($)
Engineering Services	200	18	3600
Test & Product Support Engineering	332	23	7636
Materials	21	26	546
Quality Control	18	26	468
Advance Manufacturing	24	26	624
		TOTAL COST:	12,874

EQUIPMENT COST

GenRad Fixture:	7000
Manufacturing Aids:	1000
Test Equipment:	7000
Total Cost:	15,000

ARTWORK REVISION COST

Given Current A/W with

15 Blue Lines to

Obtain New A/W

Total Cost: 1000

Table 5-2: Cost to introduce (initial set-up costs) estimates for VAD system as currently implemented

The preliminary architecture for the redesigned VAD system is shown in Figure 5-1. The system consists of two identical, independent TMS32020-based modules. Included in each module is an analog input section, A/D/A conversion, serial bitstreaming circuitry, clock sensing circuitry, and 24K words of memory. The memory is configured as 8K words of EPROM and 16K words of data RAM. The TMS32020 executes directly from the EPROM, i.e., no program RAM is included in the design.

A synchronous serial data adapter (SSDA) chip is used to implement the bitstreaming of the coded data. A filter/codec chip is used to implement the A/D/A conversion. Both chips are easily interfaced to the TMS32020, resulting in minimal board space consumption and a simple hardware design.

5.2.2 Production Cost Estimates for VAD System Redesign

Cost estimates for production of the redesigned VAD system units utilizing new technology components are given in Tables 5-3 and 5-4. Similar to the estimates for the current VAD design, Table 5-3 details the cost per unit for various production quantities and Table 5-4 gives the cost to introduce. Again, these estimates are for the processing board only, and do not include additional packaging or switches, connectors, etc., that are not on the printed circuit board.

As seen in the tables, the cost per unit in quantities of 100 are approximately $1750 per unit. Cost to introduce is about $15,000, or $150 per unit. Thus, total cost for production of 100 units is approximately $190K (or $1900 per unit).

Production costs decrease approximately 14% to $1506 per unit in quantities of 10,000. Thus, total cost for production of 10,000 units would be $15.1 million (or $1507 per unit).

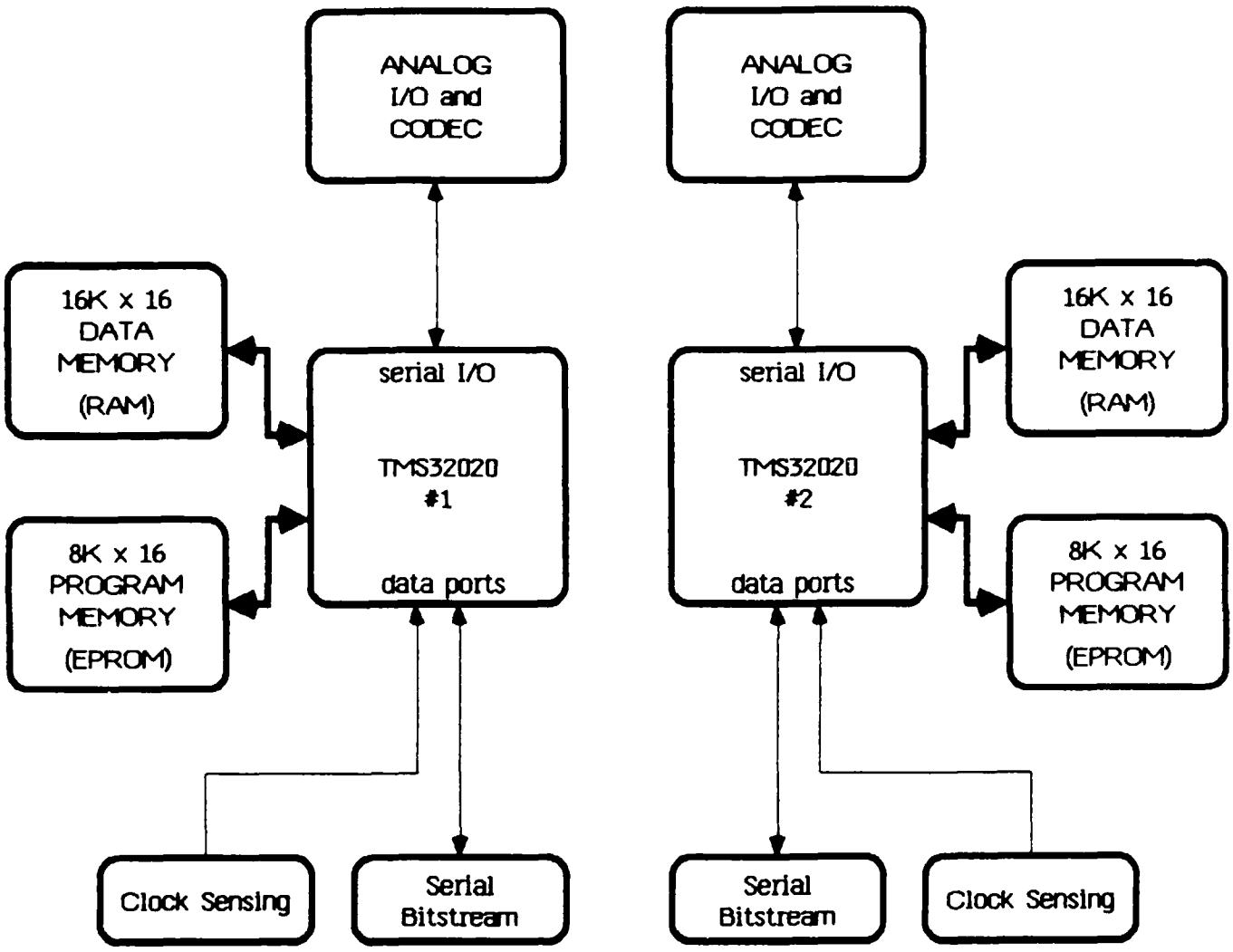


Figure 5-1: Preliminary hardware architecture for redesigned VAD system.

Product Costing	QUANTITIES			
	100	500	1000	10000
Base Material Dollars	1129 62	1070 6	1058 6	970 78
Mat l Handling at 24 4%	275 6273	261 2264	258 2984	236 8703
TOTAL MATERIAL	1405 247	1331 826	1316 898	1207 650
Labor Hours				
HK hours	2.7	2.7	2.6	2.3
$11.33/hr	30.591	30.591	29.458	26.059
US hours	2.1	2.1	2	1.8
$45.51/hr	95.571	95.571	91.02	81.918
TOTAL LABOR	126.162	126.162	120.478	107.977
SUB TOTAL	1531.409	1457.988	1437.376	1315.627
CCMD General & Admin. at 13.7%	209.8031	199.7444	196.9206	180.2409
Cost of Money				
US at $1 67986/hr	3.527706	3.527706	3.35972	3.023748
HK at $0 17048/hr	.460296	.460296	.443248	.392104
CCMD G & A at 0.00496/$	7.595790	7.231622	7.129387	6.525512
TOTAL PRODUCT COST	1752.796	1668.952	1645.229	1505.810

Table 5-3: Per unit production cost estimates for the redesigned VAD system.

	HRS	RATE($)	COST($)
Engineering Services	100	18	1800
Test & Product Support Engineering	163	23	3749
Materials	11	26	286
Quality Control	9	26	234
Advance Manufacturing	24	26	624
		TOTAL COST:	6693

EQUIPMENT COST

GenRad Fixture:	3500
Manufacturing Aids:	500
Test Equipment:	3500
Total Cost:	7500

Table 5-4: Cost to introduce (initial set-up costs) estimates for the redesigned VAD system.

5.3 Discussion of VAD Implementation Utilizing Custom and Semi-Custom Integrated Circuit Components

In many applications, semi-custom gate arrays or full-custom integrated circuits can be used to replace many of the standard LSI and MSI circuits in a hardware design. This replacement can result in a reduction in the cost, size, and power consumption and or improve the reliability of a printed circuit design

The cost of developing a semi-custom or custom IC depends largely upon the complexity of the design. For semi-custom designs it can range from $18K to $40K for a 1000-gate array; full-custom designs typically start around $40K. Although the startup cost of such an effort is high, it can still be cost-effective if a large number of standard IC's can be replaced and if large volume production is expected. In addition to reducing the cost, size, and power consumption, using one semi-custom or custom device in place of many standard IC's also reduces the bookkeeping and quality control requirements per unit.

We evaluated the feasibility of using semi-custom or custom devices in the redesigned VAD system. Many of the IC's in the preliminary redesigned VAD system in Figure 5-1 are specialized, complex components, or memories (i.e., TMS32020, CODEC, large memories, etc.). These parts would be virtually impossible to integrate into a gate-array because of their complexity. A full-custom design incorporating more than one of these parts would also be extremely difficult. This is verified in part by the unavailability of such parts in the market (a TMS32020 with additional on-chip memory, or an on-board CODEC, for example).

A practical approach would be to keep the complex components in the design and replace the remaining circuitry with a semi-custom or custom IC. There are very few MSI and LSI parts in the preliminary VAD redesign. Most of these IC's would easily fit in a gate array. However, even assuming the lowest development cost for semi-custom devices, it would not become cost-effective to replace these parts until the volume reached the 10,000 unit range. Unless such high-production volumes are anticipated, the redesigned VAD system should be implemented using MSI logic and the specialized IC's discussed above.

6. REFERENCES

1. V. Viswanathan, W. Russell, and A. Higgins, "Design and Real-Time Implementation of a Robust APC Coder for Speech Transmission over 16 kb/s Noisy Channels", Final Report, Vol. I: Algorithm Design and Simulation, Report No. 4565, Bolt Beranek and Newman Inc., December 1980, Contract No. DCA100-79-C-0037, AD-A096092.

2. J. Wolf, K. Field, and W. Russell, "Design and Real-Time Implementation of a Robust APC Coder for Speech Transmission Over 16 kb/s Noisy Channels", Final Report, Vol. II: Real Time Implementation, Report No. 4565, Bolt Beranek and Newman Inc., December 1980, Contract No. DCA100-79-C-0037, AD-A096092.

3. J. Wolf and K. Field, "Real-Time Speech Coder Implementation on an Array Processor", *IEEE Trans. Commun.*, Vol. COM-30, No. 4, April 1982, pp. 615-620.

4. V. Viswanathan, M. Berouti, A. Higgins, and W. Russell, "Development of a Good-Quality Speech Coder for Transmission Over Noisy Channels at 2.4 kb/s", Final Report No. 4916, Bolt Beranek and Newman Inc., March 1982, AD-A114068.

5. V.R. Viswanathan and W.H. Russell, "Subjective and Objective Evaluation of Pitch Extractors for LPC and Harmonic Deviations Vocoders", Final Report No. 5726, Bolt Beranek and Newman Inc., July 1984, Contract No. DCA100-83-C-0023.

6. L.K. Cosell, A.G. Derr, K.D. Field, and J.M. Tiao, "Hardware and Software Documentation for High Quality 16 KBit/s Voice A/D Implementation", Technical Report 6206, Bolt Beranek and Newman Inc., April 1986.

END

DTIC

11-86